Pipeline Data Hazards

Dealing With Data Hazards

- In Software
  - 
- In Hardware
  - 

Data Hazards are caused by *instruction dependences*. For example, the add is data-dependent on the subtract:

  - subi $5, $4, #45
  - add $8, $5, $2

Dealing with Data Hazards in Software

How Many No-ops?

- sub $2, $1, $3
- subi $5, $4, #45
- add $8, $5, $2
- slt $1, $6, $7
- and $12, $2, $5
Are No-ops Really Necessary?

sub $2, $1,$3
and $4, $2,$5
or $8, $3,$6
add $9, $2,$8
slt $1, $6,$7

Dealing with Data Hazards in Hardware
Part II-Pipeline Stalls

sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)

Pipeline Stalls

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

add $12, $3, $5
or $13, $6, $2
add $14, $12, $2
sw $14, 100($2)

Pipeline Stalls

- To ensure proper pipelined execution in light of register dependences, we must:
  - detect the hazard
  - stall the pipeline
Knowing When to Stall

- 6 types of data hazards
  - two reg reads * 3 reg writes

Stalling the Pipeline

- Once we detect a hazard, then we have to be able to stall the pipeline (insert a bubble).
- Stalling the pipeline is accomplished by
  - (1) preventing the IF and ID stages from making progress
    - the ID stage because it cannot proceed until the dependent instruction completes
    - the IF stage because we do not want to lose any instructions.
  - (2) essentially, inserting “nops” in hardware

The Pipeline

- What comparisons tell us when to stall?

Stalling the Pipeline

- Preventing the IF and ID stages from proceeding
  - don’t write the PC (PCWrite = 0)
  - don’t rewrite IF/ID register (IF/IDWrite = 0)
- Inserting “nops”
  - set all control signals propagating to EX/MEM/WB to zero
Reducing Data Hazards Through Forwarding

- The Previous Data Path handles two types of data hazards
  - EX hazard
  - MEM hazard
- We assume the register file handles the third (WB hazard)
  - if the register file is asked to read and write the same register in the same cycle, we assume that the reg file allows the write data to be forwarded to the output
Eliminating Data Hazards via Forwarding

\begin{align*}
\text{sub } & $2, $1, $3 \\
\text{and } & $6, $2, $5 \\
or & $13, $6, $2 \\
\text{add } & $14, $2, $2 \\
\text{sw } & $15, 100($2) \\
\end{align*}

CSE 141, S2'06

Jeff Brown

Forwarding in Action

Instruction Fetch

\begin{align*}
\text{add } & $1, $12, $3 \\
\text{sub } & $12, $3, $4 \\
\end{align*}

Instruction Decode

\begin{align*}
\text{add } & $3, $10, $11 \\
\end{align*}

Write Back

CSE 141, S2'06

Jeff Brown
Eliminating Data Hazards via Forwarding??

CSE 141, S2'06

Try this one...

Show stalls and forwarding for this code

add $3, $2, $1
lw $4, 100($3)
and $6, $4, $3
sub $7, $6, $2
add $9, $3, $6

CSE 141, S2'06

Eliminating Data Hazards via Forwarding and stalling

CSE 141, S2'06

Datapath with Hazard-Detection

CSE 141, S2'06

if (ID/EX.MemRead and ((ID/EX.RegisterRt = IF/ID.RegisterRs) or (ID/EX.RegisterRt = IF/ID.RegisterRs))) then stall the pipeline

Jeff Brown

Jeff Brown
Data Hazard Key Points

- Pipelining provides high throughput, but does not handle data dependences easily.
- Data dependences cause data hazards.
- Data hazards can be solved by:
  - software (nops)
  - hardware stalling
  - hardware forwarding
- Our processor, and indeed all modern processors, use a combination of forwarding and stalling.
- ET = IC * CPI * CT